Noise-contrastive estimation of normalising constants and GANs

Fonctions génériques

```
library(ggplot2)
library(reshape)
library(matrixStats)
```

Algorithme d'Hasting

Utilité : simuler selon $p_m(.,\psi)$ pour un paramètre ψ choisi.

Argument	Type	Exemple	Indication
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
n	entier	100	taille de la simulation
psi	vecteur	c(0,1)	paramètres de la fonction h
h	fonction		fonction qui retourne $\overline{p_m}(.,\psi)$

```
hasting = function(x, n, psi, h){
  y = c()
  y = append(y, sample(sample(x, 1)))
  for (i in 2:n){
    y_ = rnorm(1, y[i-1], 1)
    u = runif(1)
    if ( u <=
         (h(y_,psi) * dnorm(y_, y[i-1], 1))
         /(h(y[i-1],psi) * dnorm(y[i-1], y_, 1))
    ){
      y = append(y, y_)}
    else {
      y = append(y, y[i-1])
    }
  }
  return (y)
}
```

Note : on peut très certainement écrire sous forme matricielle cette fonction pour une meilleure performance.

MC MLE

Utilité : retourne une estimation des paramètres selon la méthode décrite dans le papier de Geyer.

```
mc_mle = function(x, psi, h){

  m = length(x)

  y = hasting(x, m, psi, h)
```

```
  L = function(theta){
    return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
  }

  theta = optim(
    par = rep(1,length(psi)),
    gr = "CG",
    control = list(fnscale=-1),
    fn = L
  )$par

  return(theta)
}
```

NCE

Utilité : Retourne l'estimation de la constante et des paramètres.

Argument	Type	Exemple	Indication
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
law_y	fonction	rnorm	fonction qui retourne un échantillon suivant la loi p_n
n	entier	100	taille de l'échantillon de bruit suivant la loi p_n
params_y	vecteur	c(0,1)	arguments de la fonction law_y
log_pm	fonction		fonction qui retourne le logarithme de la densité p_m
log_pn	fonction		fonction qui retourne le logarithme de la densité p_n
size_theta	entier	3	taille de θ, vaut habituellement 2 ou 3
method	string	"CG"	méthode d'optimisation, habituellement "CG" ou "BFGS"

```
nce = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, methode = "CG"){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  h = function(u, theta){
    return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
  }

  J = function(theta){
    return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
  }

  theta = optim(
    par = rep(1, size_theta),
    gr = methode,
    control = list(fnscale=-1),
    fn = J
  )$par

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

Graphiques

Utilité : afficher l'histogramme pour un échantillon de données x.

```
print_hist = function(x) {
  df = data.frame(x = x)
  hist_x = ggplot(df, aes(x=x)) + geom_histogram(aes(y = stat(count) / sum(count)), bins = 20, color="wh
  print(hist_x)
}
```

Utilité : pour NCE, afficher l'évolution des paramètres au fur et à mesure de l'augmentation de n (la dimension de l'échantillon de bruit)

```
NCE_evol_params = function(x, law_y, params_y, log_pm, log_pn, size_theta, ratio, steps, labels, methode

  # Creation de l'abscisse
  m = length(x)
  N = seq(0, m*ratio, length.out = steps + 1)

  # Creation de l'ordonnée
  theta = c()
  for (n in N) {
    theta = append(theta, nce(x, law_y, params_y, log_pm, log_pn, size_theta, n, methode))
  }

  # Formatage des données
  theta = t(rbind(matrix(theta, nrow = size_theta),N))
  df = as.data.frame(theta)
  df_melted = melt(df, id.vars = "N")

  # Plot
  plot_df = ggplot(df_melted, aes(x = N, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  geom_point(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par rapport au bruit", x = "n (taille du bruit)", y = "Paramèt
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))

  print(plot_df)

  return(theta)
}
```

Note : il faudrait optimiser le temps de calcul de ces fonctions, peut-être en matriciel au lieu des boucles ou bien avec du calcul en parralèle sur CPU/GPU

Exemple basique : la loi normale

Soit x l'échantillon de taille m obtenu selon la loi de densité inconnue p_d.

On considère ici que p_d appartient à la famille de fonctions paramétrées par $\theta = (c, \mu, \sigma)$ suivante :

$$p_m(u;\theta) = \frac{1}{Z(\mu,\sigma)} \times exp\big[-\frac{1}{2}\big(\frac{u-\mu}{\sigma}\big)^2\big] \quad \text{d'où} \quad ln(p_m(u;\theta)) = c - \frac{1}{2}\big(\frac{u}{\sigma} - \frac{\mu}{\sigma}\big)^2$$

```
pm_barre = function(u, theta){
  return(exp(-0.5 * ((u - theta[1]) / theta[2]) ** 2))
```

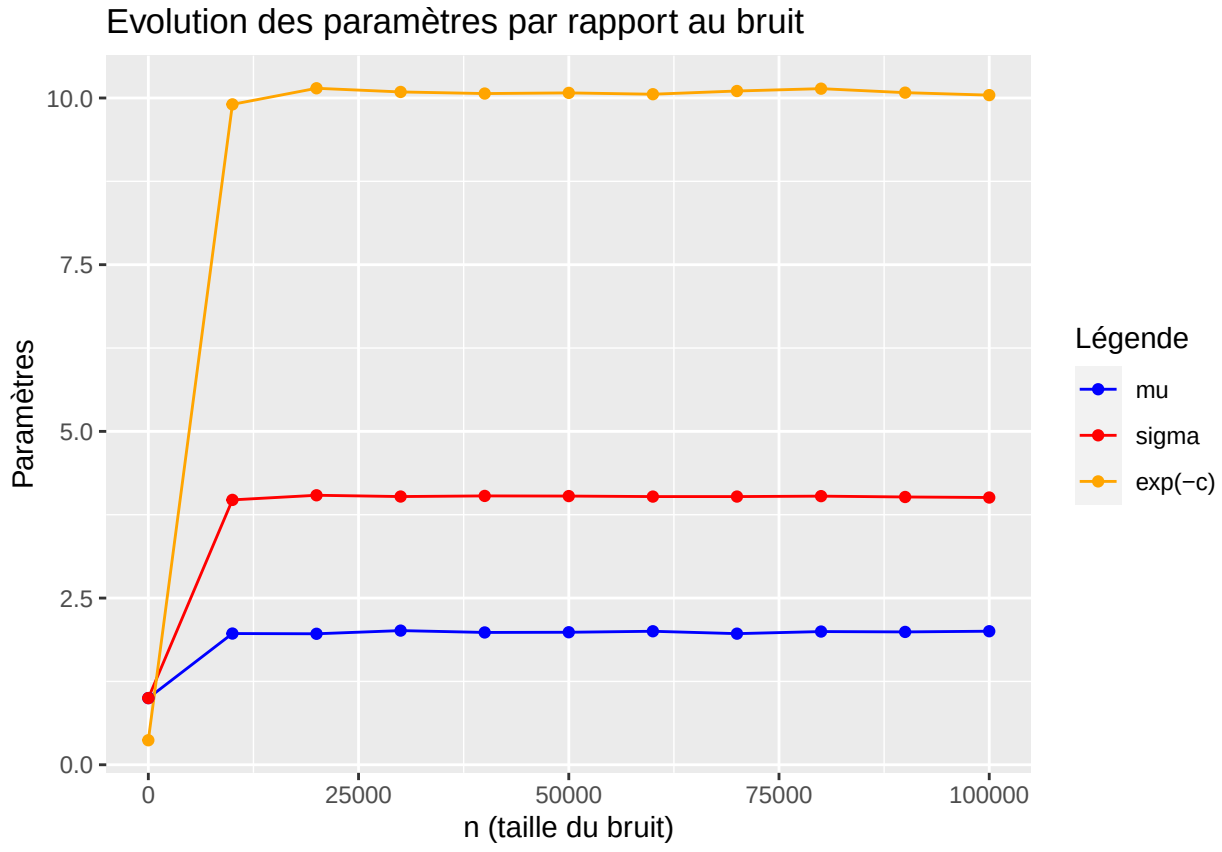
```
}

log_pm = function(u,theta){
  return(theta[3] - 1/2 * (u/theta[2] - theta[1]/theta[2]) ** 2)
  # theta[1] = mu / theta[2] = sigma / theta[3] = c
}

log_pn_cauchy = function(u){
  return(log(dcauchy(u, mean(x), sd(x))))
}

m = 10000
n = 10000
x = rnorm(m, 2, 4)
size_theta = 3

# METHODE MC MLE
mc_mle(x, c(mean(x),sd(x)), pm_barre)
```

```
## [1] 1.748345 4.400524
```

```
# METHODE GEYER
nce(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n)
```

```
## [1]  1.963537  3.997467 10.009988
```

```
NCE_evol_params(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, 10, 10, c("mu", "sigma
```



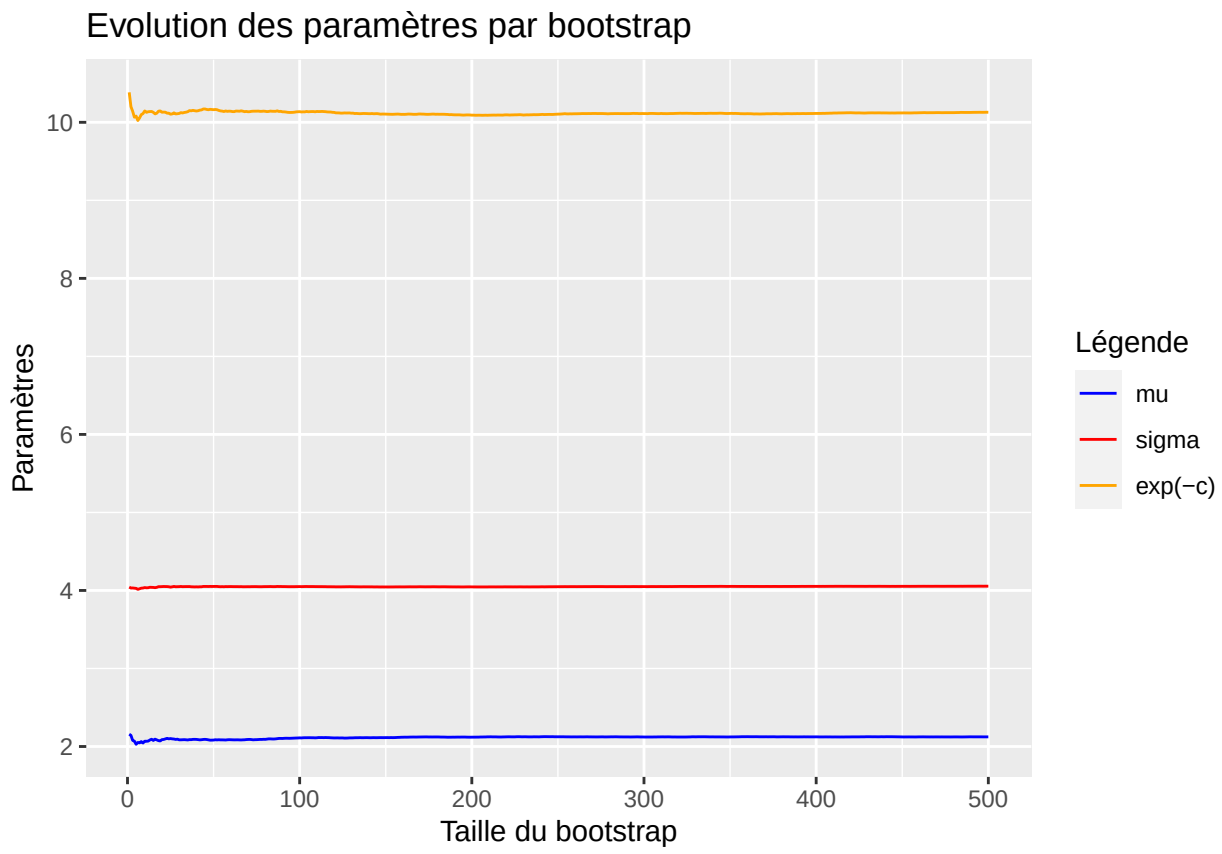
```
##                                       N
##  [1,] 1.000000 1.000000  0.3678794 0e+00
##  [2,] 1.967989 3.972487  9.9041889 1e+04
##  [3,] 1.964534 4.041750 10.1455509 2e+04
##  [4,] 2.012164 4.022650 10.0905889 3e+04
##  [5,] 1.984484 4.032058 10.0663713 4e+04
##  [6,] 1.987486 4.029703 10.0763772 5e+04
##  [7,] 2.002240 4.022022 10.0563237 6e+04
##  [8,] 1.966546 4.022355 10.1044782 7e+04
##  [9,] 1.997824 4.028628 10.1396810 8e+04
## [10,] 1.992467 4.016031 10.0798583 9e+04
## [11,] 2.002858 4.008182 10.0427378 1e+05
```

Tentative de bootstrap

Utilité : utilise le bootstrap sur x pour estimer les paramètres

```
NCE_bootstrap = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, nb_of_boot, labels, methode

  theta_bootstrap = c()
  for (i in 1:nb_of_boot) {
    x_bootstrap = sample(x, size = m, replace=TRUE)
    theta_bootstrap = append(theta_bootstrap, nce(x_bootstrap, law_y, params_y, log_pm, log_pn, size_the
  }

  # Formatage des données
  theta_bootstrap = matrix(theta_bootstrap, nrow = size_theta)

  # Plot
  size_bootstrap = 1:nb_of_boot
  df = as.data.frame(cbind(t(rowCumsums(theta_bootstrap)) / size_bootstrap, size_bootstrap))
  df_melted = melt(df, id.vars = "size_bootstrap")

  plot_df = ggplot(df_melted, aes(x = size_bootstrap, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par bootstrap", x = "Taille du bootstrap", y = "Paramètres", c
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))

  print(plot_df)

  return(rowMeans(theta_bootstrap))
}
```

```
(NCE_bootstrap(rnorm(1000,2,4), rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, 1000, 500
```




```
## [1]  2.123811  4.054793 10.128577
```

Tentative récursivité

Utilité : améliorer récursivement la précision de l'estimation via les estimations précédentes

```
mc_mle_recursif = function(x, psi, h, size_of_loop){

  m = length(x)

  for (i in 1:size_of_loop) {
    y = hasting(x, m, psi, h)

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
    }

    psi = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    print(psi)
  }
```

```
  return(psi)
}
```

```
mc_mle_recursif(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 2.221372 4.051464
## [1] 2.648556 3.900224
## [1] 2.335877 3.989864
## [1] 2.017980 3.975704
## [1] 2.416871 3.896364
## [1] 1.375132 3.848610
## [1] 1.339572 4.360124
## [1] 1.258709 4.407036
## [1] 2.387596 3.980858
## [1] 2.187077 3.797235
```

```
## [1] 2.187077 3.797235
```

```
mc_mle_recursif_2 = function(x, psi, h, size_of_loop){

  m = length(x)
  M = m

  for (i in 1:size_of_loop) {
    y = hasting(x, M, psi, h)

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - M*log(mean(h(y,theta)/h(y,psi))))
    }

    theta = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    x = append(x, hasting(x, m, theta, h))
    M = M + m

    print(theta)
  }

  return(theta)
}
```

```
mc_mle_recursif_2(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 2.165464 4.048612
## [1] 1.929229 3.869324
## [1] 2.102775 3.884618
## [1] 2.048628 3.913684
## [1] 1.991012 3.942873
## [1] 1.790131 3.919799
## [1] 1.993358 3.908191
```

```
## [1] 2.073693 3.912860
## [1] 1.983676 3.907888
## [1] 1.941447 3.844083
```

```
## [1] 1.941447 3.844083
```

```
mc_mle_recursif_3 = function(x, psi, h, size_of_loop){

  m = length(x)
  M = m
  y = hasting(x, m, psi, h)

  for (i in 1:size_of_loop) {

    L = function(theta){
      return(sum(log(h(x,theta)/h(x,psi))) - M*log(mean(h(y,theta)/h(y,psi))))
    }

    theta = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    x = append(x, hasting(x, m, theta, h))
    y = append(y, hasting(x, m, psi, h))
    M = M + m

    print(theta)
  }

  return(theta)
}
```

```
mc_mle_recursif_3(x, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 1.742960 4.052942
## [1] 1.901390 4.155794
## [1] 2.065405 4.002743
## [1] 2.12093 3.99018
## [1] 1.972605 3.981015
## [1] 1.890396 3.977155
## [1] 1.817325 3.977917
## [1] 1.848152 3.988905
## [1] 1.815458 3.952464
## [1] 1.807083 3.957232
```

```
## [1] 1.807083 3.957232
```

```
nce_recursif = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_of_loop){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)
```

```
  for (i in 1:size_of_loop){

    h = function(u, theta){
      return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
    }

    J = function(theta){
      return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
    }

    theta = optim(
      par = rep(1, size_theta),
      gr = "CG",
      control = list(fnscale=-1),
      fn = J
    )$par

    print(theta)

    y = do.call(law_y, c(list(n),theta[-size_theta]))

  }

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

```
nce_recursif(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n, 10)
```

```
## [1]  1.939518  3.996652 -2.307801
## [1]  2.051814  4.004351 -2.309591
## [1]  1.905621  4.038657 -2.316924
## [1]  2.019341  3.983065 -2.292089
## [1]  1.964496  3.987290 -2.306045
## [1]  2.063380  3.995259 -2.302239
## [1]  1.899331  4.026456 -2.318875
## [1]  2.067943  3.994603 -2.299603
## [1]  1.901653  4.005033 -2.308546
## [1]  1.953847  4.011724 -2.311482
```

```
## [1]  1.953847  4.011724 10.089366
```